SKETCHSEEKER : FINDING SIMILAR SKETCHES

A Thesis

by

JAIDEEP RAY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tracy Hammond
Co-Chair of Committee,	Aakash Tyagi
Committee Member,	Hye-Chung Kum
Head of Department,	Dilma Da Silva

May 2016

Major Subject: Computer Science

ABSTRACT

Searching is an important tool for managing and navigating the massive amounts of data available in today's information age. While new searching methods have become increasingly popular and reliable in recent years, such as image-based searching, these methods are more limited than text-based means in that they don't allow generic user input. Sketch-based searching is a method that allows users to draw generic search queries and return similar drawn images, giving more user control over their search content. In this thesis, we present Sketchseeker, a system for indexing and searching across a large number of sketches quickly based on their similarity. The system includes several stages. First, sketches are indexed according to efficient and compact sketch descriptors. Second, the query retrieval subsystem considers sketches based on shape and structure similarity. Finally, a trained support vector machine classifier provides semantic filtering, which is then combined with median filtering to return the ranked results. SketchSeeker was tested on a large set of sketches against existing sketch similarity metrics, and it shows significant improvements in both speed and accuracy when compared to existing known techniques. The focus of this thesis is to outline the general components of a sketch retrieval system to find near similar sketches in real time.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Sketching is a universal form of expression. Humans can render any object on any kind of surface using sketching, and with touch devices becoming an increasingly integral form of communication, it is important that sketch-based systems be applied to more domains. Searching is one domain where sketching is a relatively new form of input. The amount of data to search through increases every day, and while text-based querying is fast and flexible, image-based searching can be a bit limiting. Using sketch similarity, we can search across large numbers of drawings while still allowing the user maximum flexibility over their query.

In this thesis, we describe a sketch retrieval engine, SketchSeeker, for querying by sketch content. Efficient sketch indexing and querying is an important aspect of sketch-based image searching, but it is still a difficult problem to solve despite recent interest in this area by multiple search engines. This work focuses specifically on sketch indexing and querying as it relates to sketch retrieval. Sketch retrieval is a relatively new field, but it has widespread applications and can enable more sketch-based interactions. For instance, in addition to its potential applications in sketch-based image searching, it can also be used for finding clip art or indexing an enormous number of sketches in a database.

SketchSeeker is a single system which combines three distinct subsystems. The first such subsystem is the sketch-indexing stage. In this stage, we describe a method for optimizing storage of sketches in a database using a highly-compressed, searchable representation. The sketch descriptors, which consider both shape and structure, along with the compression, performed using a deep auto-encoder architecture, make retrieval significantly better in terms of time and space complexity when compared

to other known methods. The second subsystem is query retrieval. Similar sketches are returned based on their shape and structure according to their distance from the query shape in a kD-tree, as bound by an empirical threshold. The final component is a ranker, which sorts retrieved sketches based on two layers of filtering. First, semantic filtering is performed on the search results using a Support Vector Machine (SVM) classifier which returns the most likely label for each sketch. After the filtering based on most likely semantic meaning is complete, a median filter is used to eliminate any outliers before returning the final result set.

The remainder of this thesis is laid out as follows. First, we discuss the related works to SketchSeeker. Since sketch retrieval is similar to many other searching problems, and this solution relies upon many different components for portions such as sketch descriptor, semantic filtering, and ranking, there are a multitude of related works from varying domains. An additional section on sketch similarity itself follows previous works in order to motive SketchSeeker more clearly and set it apart from existing methods. Next is a more detailed description of the system and its components, divided into the three primary subsystems previously discussed. Results and a discussion are then provided to compare SketchSeeker against other similarity metrics and evaluate its retrieval performance. The thesis closes by considering future work and conclusions. There are three appendices to the thesis detailing a brief introduction and theory to SIFT, SVM based classification and Autoencoders. There are figures in the thesis which represent query retrieval results from sketchseeker. In these figures, the first sketch should be assumed to be the query and the rest are retrieved results. Figure 1.1 shows us several different classes of human drawn object sketches.

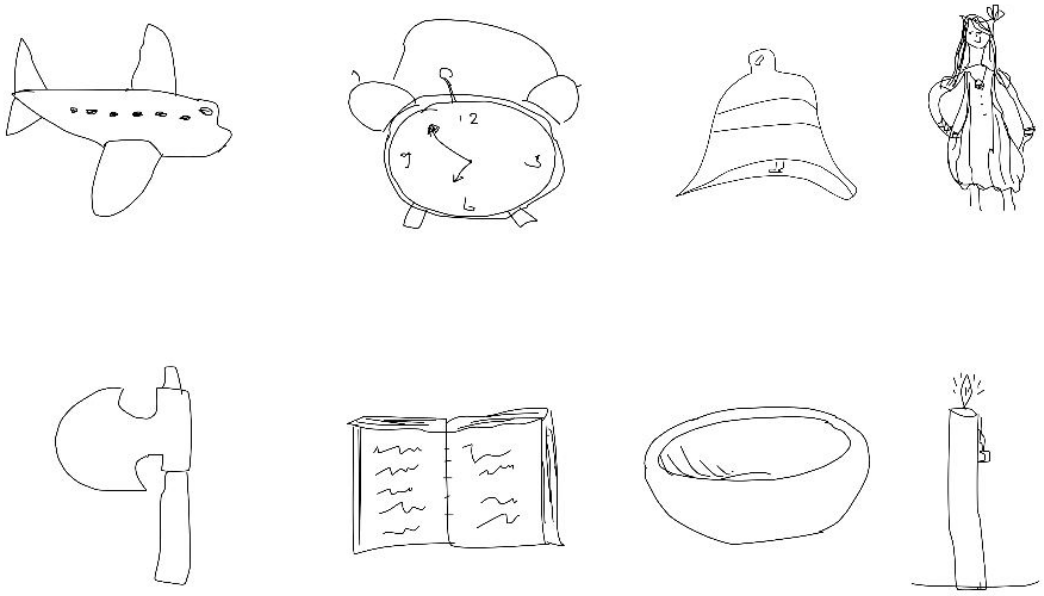

Figure 1.1: Human drawn object sketches. The dataset consists of 250 such classes of object sketches each having 80 sketch. This collection of sketches has classes airplanes, alarm clock, bell, angel, axe, book, bowl and candle.

2. RELATED WORKS

Sketch retrieval overlaps with a number of other areas. In addition to the connection with searching and classification, it relies heavily on sketch compression and representation, as well as similarity metrics. Most previous work in the area of sketch searching has been based on approximate matching using a sketch descriptor. The focus of these papers was the accuracy of the descriptor in capturing sketch features. To name just a few, these descriptors include shape context, Scale-Invariant Feature Transform (SIFT) descriptors, Hausdorff metric, and Fisher vectors. The next section provides a more directed discussion of similarity metrics and shape descriptors, but much of the relevant work related to sketch recognition, classification, categorization, and retrieval is mentioned below.

2.1 Sketch Recognition

Sketch Recognition is the algorithmic recognition of hand drawn sketches. It consists of many different algorithms, some designed for specific domains and others designed for general recognition. It is primarily feature-based recognition, but features extracted from a sketch may be visually, geometrically, or modality based. Rubine [21] features are the most well known geometric sketch features. Rubine features have been extended for sketch recognition algorithms in multiple domains. The One Dollar recognizer can detect multiple geometric shapes using corner detection algorithms and Rubine features [34]. Topological information about sketches have also been used for recognition. This approach works well with sketches having multiple spatial components like chemical bonds or complex electrical circuits [17]. Ladder [12] is a language to describe how sketched diagrams in a domain are drawn, displayed and how recognition algorithms can be written for that domain [12].

2.2 Template Matching

Template matching aims at finding similarities between a given sketched figure and the query. While template matching has very broad applications, in sketch-related domains, it has primarily been applied to geometrical shapes and engineering drawings [1],[28],[27]. While template matching can be very effective at identifying if two sketches are similar geometrically and have a set structure, it is important to remember that human object sketching is highly varied. As a result, template matching methods that work well with geometric shapes may fail in the case of free hand sketches.

2.3 Classification for Sketching

Multiple techniques for image classification have been applied to hand drawn sketches with some success. SIFT, Fisher vectors, and such descriptors have been used for sketch classification [7],[6],[10],[8],[9]. SIFT, short for Scale-Invariant Feature Transform, and Fisher vectors are techniques for encoding an image according to its gradient. Blocks of the image are described by the direction of change relative to the cardinal and intermediate directions, eight orientations in total. There have also been some attempts to do a classification-driven analysis which can predict the semantic aspects of sketch [23].

2.4 Multi-modal Search

Here, multi-modal search refers to searching by sketching across different domains. The sketched lines are taken as the input query and used to retrieve media files, images, or 3D shapes [18] [11]. While there has been some interest in multi-modal searching, sketch retrieval has not been as explored [25].

2.5 Sketch Retrieval

Sketch-to-sketch retrieval is a known problem but has not been well-explored. The popularity of stylus-based and touch devices have made sketching an important means of digital communication, further emphasizing the importance of sketch understanding in domains like search. Sketch matching and retrieval has various applications such as trademark logo matching [22], handwriting recognition, or general searching on sketch-based interfaces or general stylus-based surfaces. Sketch retrieval according to shape topic has been employed in Mindfinder [26] [33].

2.6 Sketch Representation

Also relevant to sketch retrieval is the representation method of a sketch. Sketches can be very large to store and search through when considering all of the raw point data, so for retrieval, it is important to have a representation that balances compression with the complexity needed to accurately represent a sketch. Methods like SIFT descriptors [14] and Fisher vectors [24] are forms of representing sketches according to different kernels. Further, shape context descriptors, introduced in [2], have been shown to capture the shape information of a sketch effectively. While all of these generate a searchable representation of sketches, such descriptors are huge matrices, and it is difficult to work with them in their original form. An approach to compress shape context descriptors for efficient shape matching has been discussed in [2]. In [16], Oltmans introduces a new vision-inspired feature for representing sketches. This "Bullseye" feature uses a sliding window algorithm to generate a circular histogram of an image which may be used for match finding.

3. SKETCH SIMILARITY

A sketch retrieval system must have an indexing and a matching framework. Indexing involves generating a compressed sketch representation, while matching relies on some similarity metric to find the nearest sketches. In this section, we provide a more detailed discussion of some of the indexing and matching methods relevant to SketchSeeker.

Ideally, the indexing framework of a retrieval system will allow for a very compressed representation of a sketch, while still maintaining the ability to effectively match. The matching framework defines similarity between two sketches and should be symmetric.

$$dist(A, B) = dist(B, A)$$

$$dist(A, A) = 0$$

3.1 Hausdorff Distance

One common method for evaluating how well sketch A matches sketch B is Hausdorff distance [15], a cost metric defined by the following two equations:

$$D_A = \min_{b \in P_B}(|a - b|), a \in P_a$$

$$H_d(A, B) = max(max(D_a), max(D_b))$$

Hausdorff distance measures how far the shapes are from being isometric. Here $P_A \& P_B$ represent the cloud of points for sketch A and sketch B. D_A represents the min distance from a point in P_A to a point in P_B. Hausdorff metric tries to capture the similarity between shapes of two cloud of points. It considers each sketch as a

cloud of points, which we achieve by obtaining about 100 samples of points randomly from the sketched drawings. Unfortunately, Hausdorff is a pairwise match evaluator, so to use it within a retrieval system's matching framework, it must scan through all the sketches to find the best match. This is computationally expensive, time consuming, and inefficient in terms of storage. Furthermore, Hausdorff distance is not very stable regarding outliers or noise. This is an important factor since we are dealing with user drawn sketches of natural objects.

3.2 Feature Descriptors

A more feasible approach to indexing and matching is based on feature descriptors. Such descriptors are widely used in computer vision problems and image matching, so we should consider them when matching sketches. Feature descriptors encode certain features about a sketch like topological information of the sketch (connectedness amongst components), geometry (shape), or local features like corners or curvature.

It is a challenging task to detect what kinds of features will work best for matching sketches. Certain feature descriptors are invariant to scale, rotation or affine transformations and may work really well for matching human sketches of everyday objects. The key issues are detecting the feature points, encoding them effectively, and matching them, which may not correspond directly with techniques used for images. Much work has been done recently on content-based image retrieval [5], but images have rich description, color, texture, and shape whereas sketches have shape, temporal, and spatial stroke information only. We use adapted feature descriptors for SketchSeeker.

Image feature descriptors can be broadly classified into three categories:

1. Geometric Geometric feature descriptors try to capture the shape related in-

formation of an image, e.g. corners and sharp curves. There are well known feature descriptors in this area like Histogram of Gradient (HOG), SIFT, etc. Geometric feature descriptors have often been used in object categorization.

2. Hash-based This is also known as geometric hashing where small image patches from different parts of the image are hashed into numbers. If there is a high match between such local patches of images determined by matching the hashes, then they must have similar visual content.

3. Semantic-based Semantic hashing builds feature descriptors based on the data. It uses deep neural network to encode similar images to a similar compressed representation. Small images are compressed to short binary codes using deep auto-encoding.

Feature descriptors can also be classified according to whether they are global or locally. Local features describe a localized region of a sketch whereas global features describe something about the entire sketch.

3.3 Shape Context Descriptors

Shape context is a geometric, global feature descriptor. Shapes are very discriminative features of sketches, making this a useful descriptor in terms of indexing and matching sketches. Shape context feature descriptors are constructed using a radial-based window which constructs a histogram based on which regions of the sketch are in the radial bins [2] [16]. This method of construction is largely resistant to noise and outliers.

SketchSeeker uses shape context descriptors as just one component of sketch matching. Because these feature descriptors can also consume a lot of space when stored for a large number of sketches, we also work with a highly-compressed version

of shape context. Figure 3.1 shows the shape context description of two sketches, and Figures 3.2 and 3.3 show the calculated costs and similarity of matching the sketches.

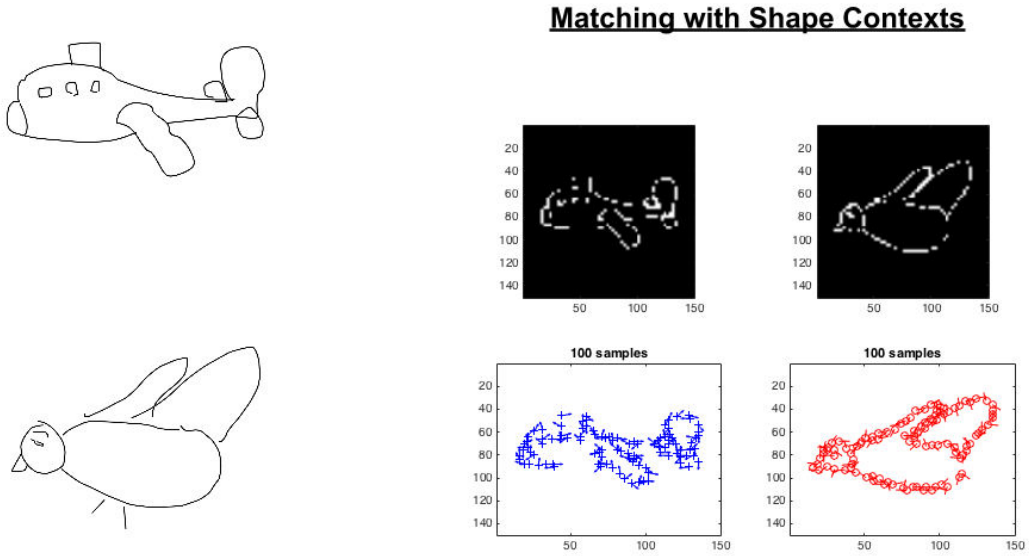

Figure 3.1: Shape context descriptors extraction; figure shows how a sketch is sampled into a cloud of points before feature extraction.

3.4 SIFT Feature Descriptors

For reliable description or for tasks like recognition or retrieval, it is important that the features extracted from an input are invariant to scale change, small noise, or small affine transformations. SIFT is one such algorithm from computer vision which has been widely used for object recognition [19]. SIFT is geometric and local and attempts to extract key points of an input, for instance, corners in a sketch. Figure 3.4 shows the SIFT key-point descriptor matching.

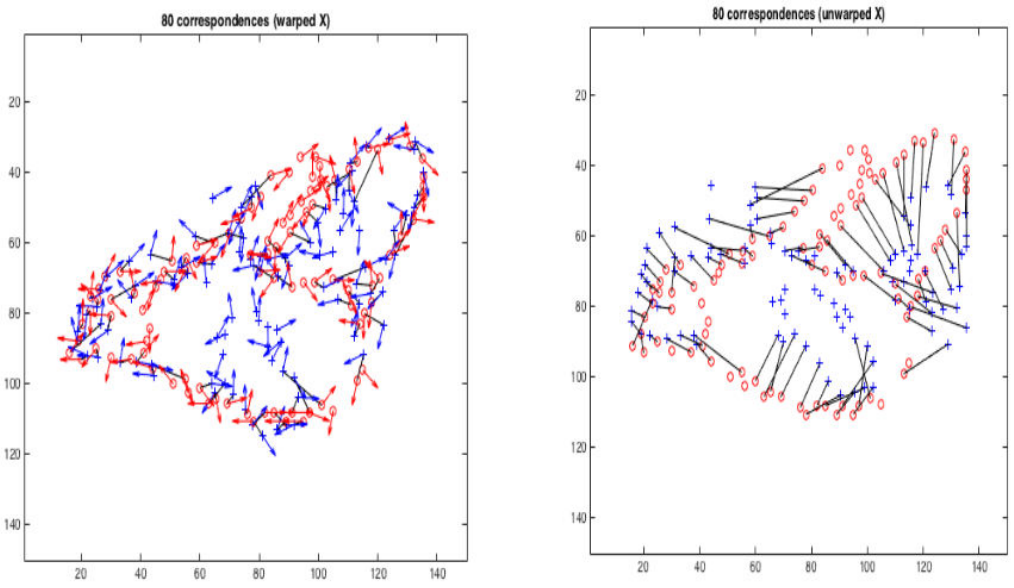

Figure 3.2: Shape context matching cost as described in [3]

One of the successful uses of SIFT features for sketch recognition has been done by Eitz et al [6]. They extracted SIFT features from sketches and trained a support vector machine (SVM) classifier to categorize sketches. SketchSeeker has a similar approach, using SIFT as one portion of the indexing stage and an SVM classifier for semantic understanding during the ranking stage. SIFT is highly compressed for SketchSeeker and paired with shape context to give more information about the input sketch; by splitting SVM classification into a single stage of the ranker, ranking can be performed considering other metrics. Figure 3.5 shows us the inter class similarity on object sketch category legs and arms. Figure 3.6 depicts the intra class similarity in airplanes.

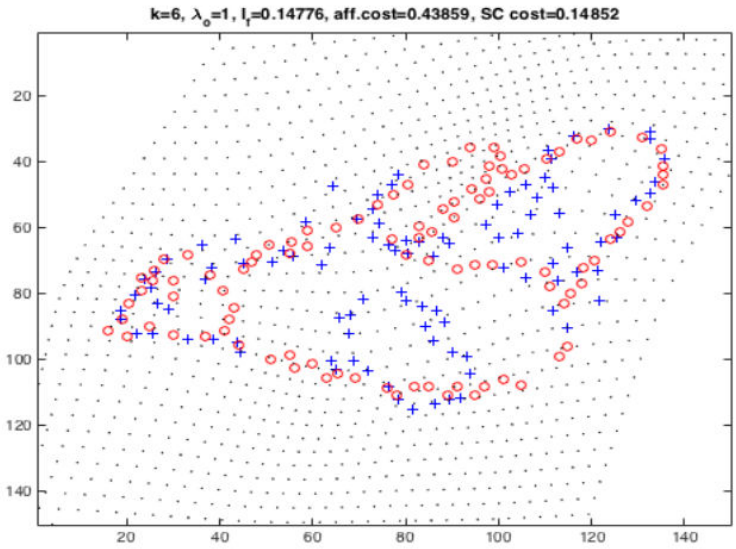

Figure 3.3: Sketch similarity by shape context matching as described in [3]

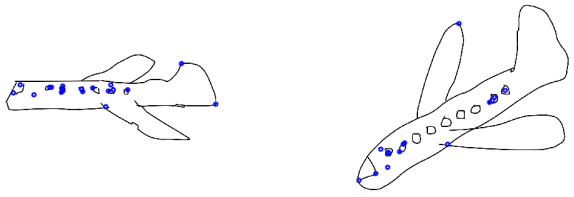

Figure 3.4: SIFT key-point descriptor matching. The blue regions have matched keypoints.

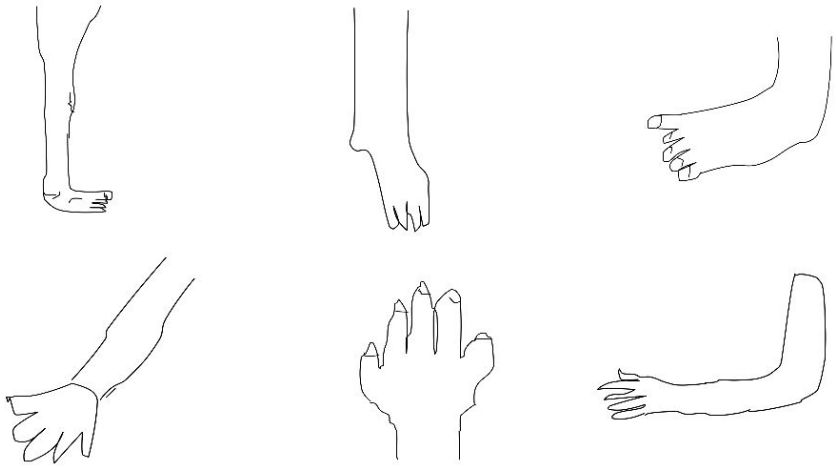

Figure 3.5: Inter-class similarity on legs and arms. Top - Legs and feet, Bottom - Arms and hands

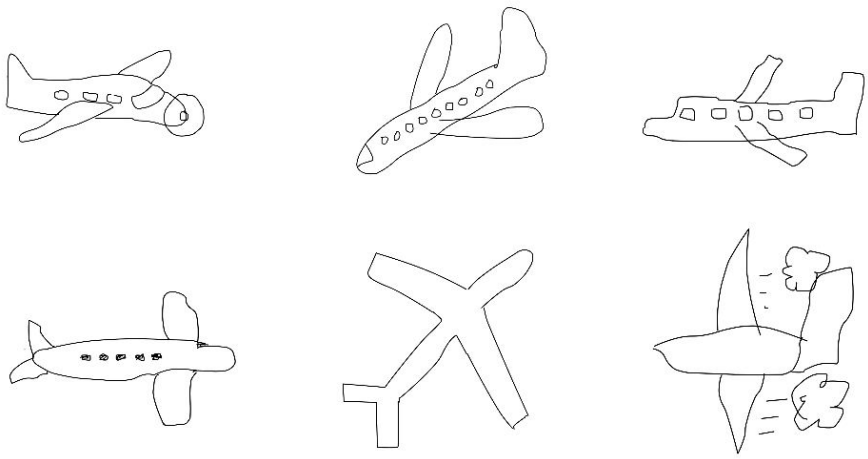

Figure 3.6: Intra-class variation in airplanes

4. METHODOLOGY

4.1 Problem Formulation

The problem is that, given an input query sketch, we must retrieve a collection of very similar sketches from the database. Similarity is defined according to both shape and semantic meaning, what the object is intended to represent. Queries should be resolved quickly, and the database representation should be as storage-efficient as possible.

4.2 Sketch Dataset

SketchSeeker uses a dataset consisting of 20,000 human drawn sketches of everyday objects. The dataset consists of 250 classes each having 80 sketches. As seen in Figure 3.5, there is some ambiguity introduced by the similarity of the objects being represented, referred to as interclass similarity. Futhermore, humans draw diversely while drawing the same object which causes intraclass variation; see Figure 3.6 for an examples with the airplane class. Sketches can also be noisy. We use a dataset of 20,000 sketches to capture as much variance, similarity, and noise as possible so that SketchSeeker can be made invariant to scale and noise while still maintaining discriminative capabilities between objects.

5. IMPLEMENTATION DETAILS

SketchSeeker accepts an input query and returns a set of nearest similar sketches from the database. It splits the task into three primary components - Indexing, Retrieval, and Ranking. Each stage is addressed in greater detail in this section.

First, sketch images are indexed in the database. This process involves computing feature descriptors and substantially compressing the representation for storage. Second, retrieval is performed by finding the nearest neighbors to a query sketch in a searchable, multi-dimensional map of the sketches in the form of a kD-tree. Third, the ranker considers the results based on their shape context and SIFT matches, along with semantic meaning as determined by an SVM classifier. Finally, the ranked result set is provided as output to the user. Figure 5.1 shows the sketch seeker design.

5.1 Indexing

5.1.1 Introduction

During indexing, we compute compact codes to represent a sketch. Making sketch representations, also called sketch signatures, heavily compressed while still maintaining enough complexity to be searched effectively is the most significant aspect of indexing. Compression speeds up the matching process, yielding faster queries, and saves storage space, but because we try to match the sketch signature of a query sketch to the sketch signatures of sketches in the database, severe compression can reduce accuracy. The similarity between two such codes or sketch signatures should represent the similarity between sketch descriptors of the corresponding sketches. SketchSeeker attains massive compression levels by combining both and shape context and SIFT information into the signature, which is compacted with minimal data

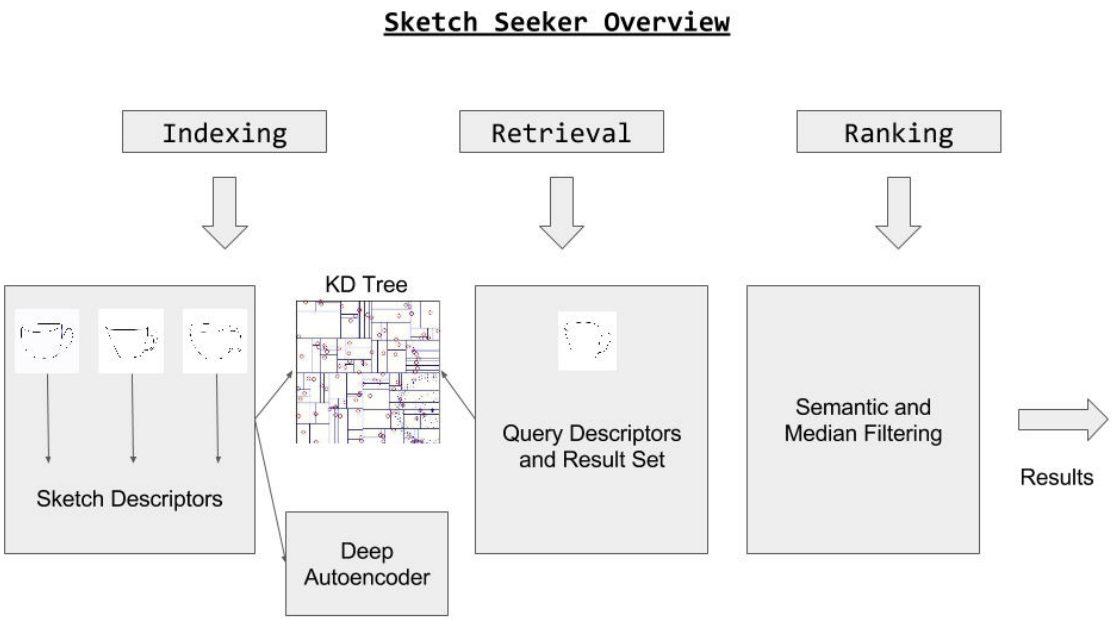

Figure 5.1: Sketch seeker design

loss by a deep auto-encoder.

5.1.2 Feature Extraction

A sketch is made up of multiple strokes. To describe a sketch completely we should consider both its appearance, e.g. shape, and the important stroke features which are captured in key-points like corners. SketchSeeker uses shape context descriptors to represent the overall shape of a sketch. To represent stroke features, we use SIFT descriptors. During the indexing process, these features are extracted from each sketch in the database, which can be done in parallel to boost efficiency of the system.

5.1.3 Pre-processing

We pre-process the sketch descriptors by vector quantization. We first sample the sketches to 100 points. Shape context descriptors are 60 dimensional vectors,

and since each sketch has 100 such descriptors, it becomes a 100 x 60 matrix. SIFT descriptors are 128 dimensional vectors, and each sketch has a different number of SIFT key-points depending on the strokes. So, SIFT descriptors are (number of key-points) x 128 matrix. We consider around 5,000 randomly-chosen SIFT key-point descriptors as the candidate set to build a vocabulary, discussed in the next step. SIFT features are extracted using the VLfeat library [32]. We vertically concatenate the descriptors from all the sketches to form a composite shape context matrix and SIFT context matrix.

5.1.4 Vocabulary Building and Binning

To compress the sketch descriptors we build a separate vocabulary for each kind of descriptor and express the descriptors in terms of this vocabulary. Vocabulary building involves clustering all the descriptors and forming a vocabulary of all chosen cluster centers. The next step involves assigning the descriptor to an index of the cluster center to which they belong; we use approximate nearest neighbor k-means to cluster the descriptors. After vocabulary building, we need to encode the descriptors with respect to this vocabulary. In the case of SIFT, where the number of key-points varies from sketch to sketch, we need to bin the sketch descriptor to get uniform size representation. Figure 5.2 outlines the steps involved in descriptor quantization and encoding. The detailed algorithm to determine the descriptor vocabulary is as follows:

1. Sample s points from the sketch.

2. Determine the feature descriptors of the sketch using these s points. Each shape context descriptor is a n dimensional vector ($n = 60$). Each SIFT descriptor is an m dimensional vector ($m = 128$). Binning is performed to make the number

of descriptors equal for each sketch. The sketch is represented as a collection of all feature descriptors.

3. Now, we perform an Approximate Nearest Neighbor (ANN) k-means clustering to determine cluster centers and cluster formations of the shape context vectors.

4. The number of cluster centers C determines the size of the vocabulary. The shape context vectors are assigned the index of the closest cluster center.

5. We now represent each sketch as a collection of shape context descriptors which are just labels of cluster centers (integers from $1, 2, .., C$).

6. The shape context descriptor becomes just a single histogram of cluster label frequencies. These frequency histograms have length $|C|$.

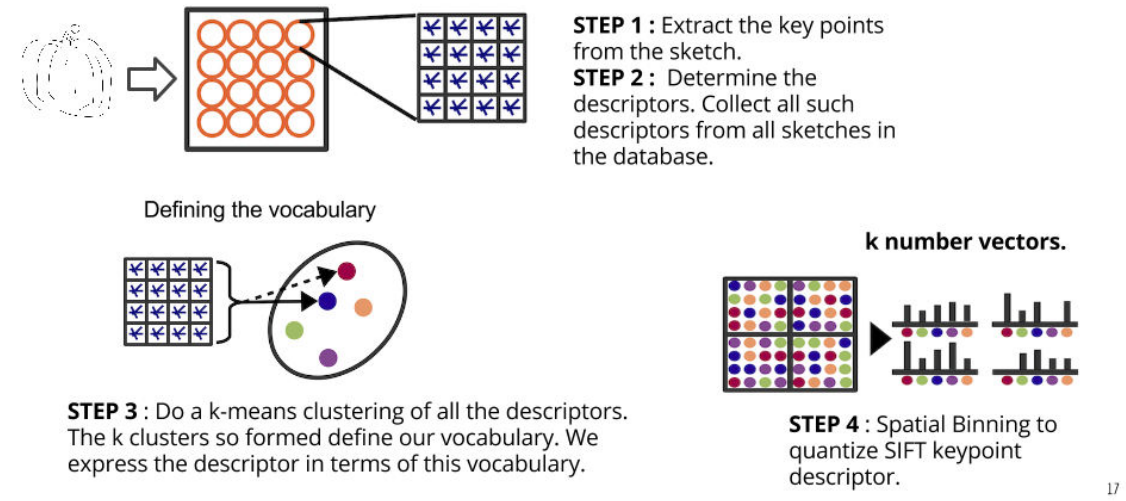

Figure 5.2: Descriptor quantization and encoding

We used a value of $C = 600$, for our experiment. A value of 600 makes the representation space-intensive and a large data-set of sketches cannot be stored in-memory with this representation.

5.1.5 Auto-encoding

We use a deep auto-encoder from the Dimensionality Reduction Toolbox [29] to learn compressed representations of the quantized shape contexts. A deep auto-encoder is a composition of two symmetrical neural networks, where the first half represents the encoding layer and the second half represents the decoding layer. The layers are restricted Boltzmann machines. In our case, the input to the deep auto-encoder is a C-length vector, and $C = 600$ for SketchSeeker. A sample encoding layer would look like:

$$600 \rightarrow 780 \rightarrow 450 \rightarrow 200 \rightarrow 10$$

The auto-encoder learns a vector of 10 numbers from a 600 length shape context descriptor. This vector is the encoded version of input. The second part is performing the opposite of this network and decodes the input from this encoded version of 10 numbers.

The detailed encoder steps are listed below.

1. The deep auto-encoders compress the shape context descriptors (size C) into a vector of p floating numbers where $p << C$. We have tested our implementation with $p = 10, 16$.

2. The deep neural network is stored to be used in the query retrieval stage.

3. After encoding, each sketch is represented by a vector of p floating point numbers. This representation serves as the signature/index to the sketch.

4. We store all such signatures in another kD-tree for efficient look-ups of nearest neighbors.

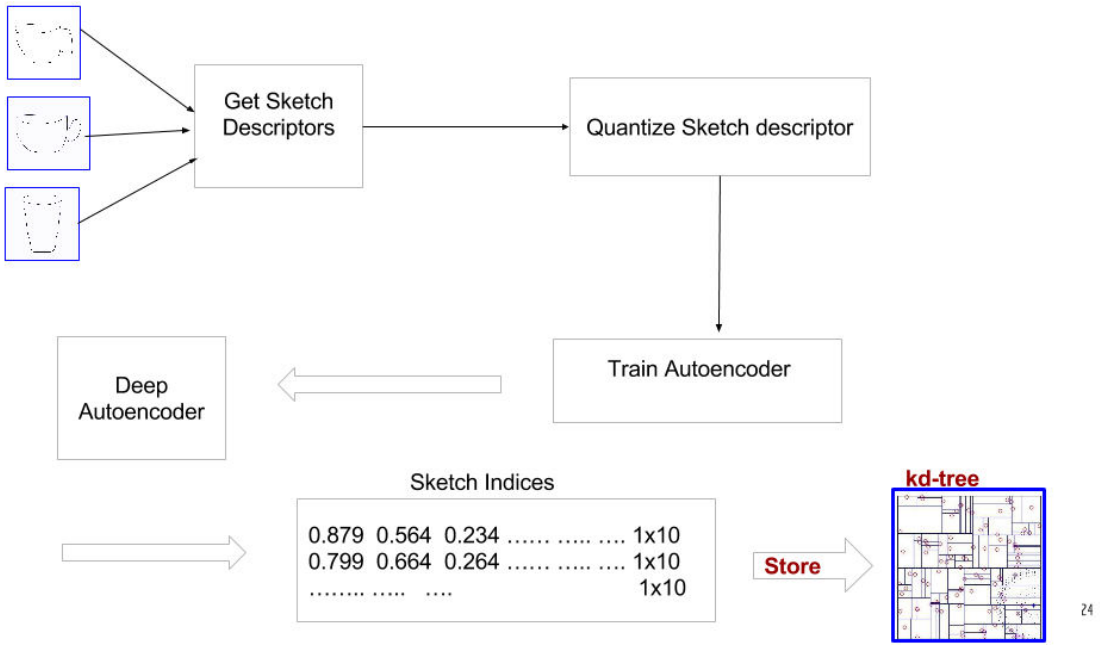

Figure 5.3: Indexing

The main idea behind learning these codes is that similar codes correspond to similar sketches. Figure 5.3 shows the entire indexing pipeline including the autoencoding.

5.1.6 Efficiency

The data structure used to store the signatures should be efficient in time and space for the retrieval of nearest neighbors to a given query. We use a kD-tree to store all the sketch signatures.

In terms of timing complexity, indexing is essentially an offline process, so it does not affect real time retrieval latencies. Still, indexing performance is vital for large

sketch databases. The major steps contributing to time complexity are the vocabulary building and auto-encoder training. The vocabulary building is dependent on the runtime of the approximate nearest neighbor k-means clustering algorithm. It is also dependent on the vocabulary size and max number of iterations allowed for convergence, which is 600 in SketchSeeker's case.

Regarding storage space, we need space proportional to the number of sketches. Because each sketch is represented by 10 numbers, SketchSeeker makes it possible to store very large sketch databases completely in-memory. Without this level of compression, the database could require an enormous amount of storage space. The Hausdorff metric must store all the stroke points. Shape context requires the number of stroke points multiplied by 60 descriptors per sketch, and as mentioned before, SIFT varies in size based on the number of key-points. Each key-point will require 128 numbers. Clearly, SketchSeeker's method makes significant gains in terms of storage efficiency through its multi-layered indexing approach.

5.1.7 Sketch Clustering Visualization

Before final evaluation, we also performed a visual analysis of our sketch descriptors to ensure their validity. We used clustering by plotting the descriptors on a 2D plane and visually checking if similar sketches are clustered together and dissimilar ones are plotted far apart. This can be done by reducing the dimensionality of the sketch descriptors using t-SNE [30] to only two dimensions and plotting the points. This exercise was performed for both sketch descriptors, and Figure 5.4 displays the results of clustering the cup dataset.

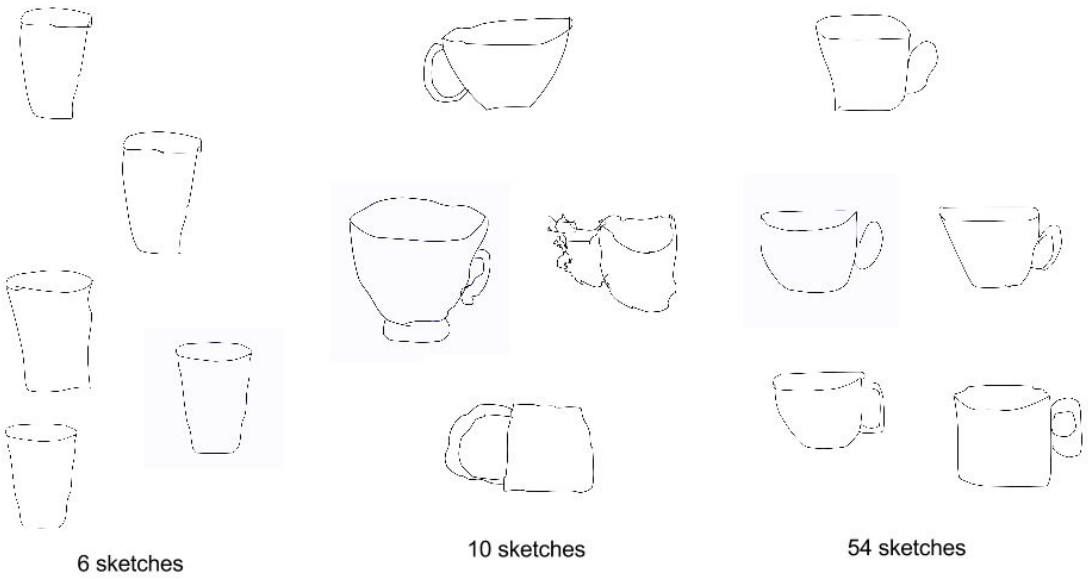

Figure 5.4: Sketch clustering - a few representative sketches from each cluster are shown.

5.2 Query Formulation and Retrieval

5.2.1 Introduction

The next major stage of SketchSeeker is query retrieval. This requires indexing the queried sketch according to the indexing stage so that similar matches can be found in the kD-tree.

5.2.2 Pre-processing

We compute the shape context descriptor and SIFT descriptor for the query sketch. We use the same pre-processing steps to quantize the query sketch as before so that it becomes a sketch signature that may be compared with the stored sketches.

5.2.3 Query-formulation

We form two queries, one based on shape context descriptor and another based on SIFT descriptors. The first tries to seek out sketches which are similar in outer shape and the second one searches sketches which have similar internal structure. These two queries can be fired in parallel.

5.2.4 Retrieval

We use the queries to search for the nearest neighbors in the kD-tree containing all sketch signatures based on shape context descriptor and SIFT descriptor. Now, we have two sets of retrieved sketches based on the descriptors.

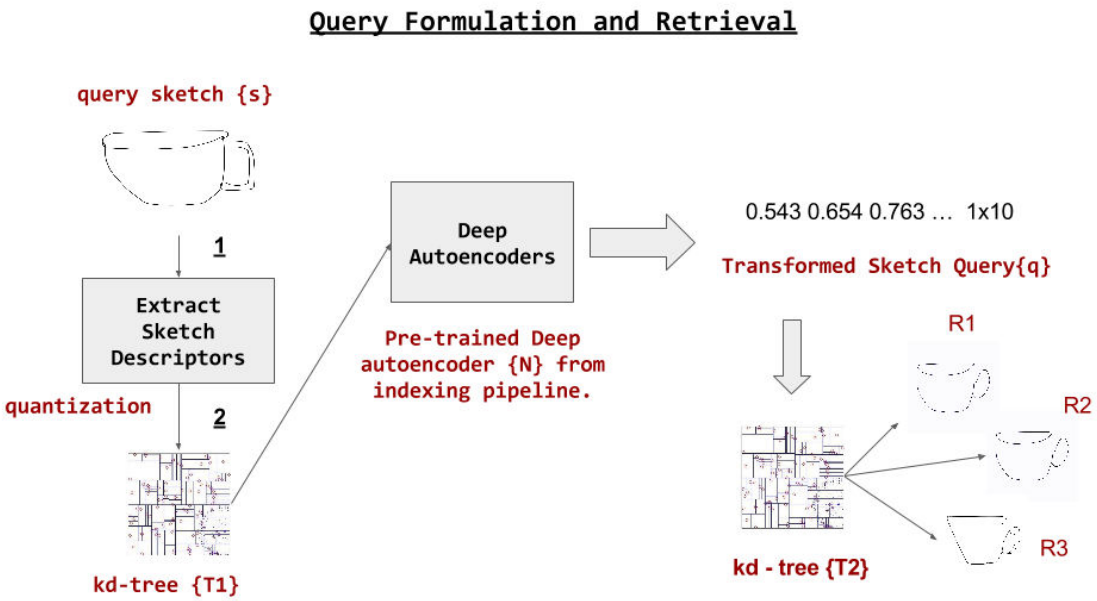

Figure 5.5: Query retrieval

The detailed steps are below.

1. Extract the shape descriptors and SIFT descriptors from the input sketch and quantize it using the method mentioned in the indexing section. It will be a constant time one-time look-up operation since we already have the kD-tree with all cluster centers in memory.

2. Use the trained auto-encoder to get the compressed representation of the quantized shape descriptor.

3. This compressed representation will serve as the signature of the sketch and our query.

4. We search for n nearest neighbors of the query sketch in the shape context kD-tree and SIFT kD-tree. This number n' is defined empirically. We experimented with $n = 25$.

5. We use both these result sets as input to the ranker to form our final result set.

The nearest neighbors are sketches which are structurally similar to the query sketch as the descriptors cover both outer shape and internal key-points. But structure cannot be the only parameter for retrieving sketches. User hand drawn sketches have a semantic meaning associated with them. They may be sketches of commonly used objects, animate or inanimate entities. Apart from searching by shape, we need to factor in the semantic meaning of sketches to have better precision. We pass on the retrieved results to the ranker in order to get the final merged resultant set. The query results also have some meta-data information – the descriptor name and corresponding distance to the query sketch based on that descriptor. Figure 5.5 shows the retrieval steps.

5.3 Ranker

5.3.1 Introduction

We designed our ranker system to rank and filter the nearest neighbor sketches to the query. We filter the result set by determining the object category of the query sketch, which is determined by a trained SVM classifier. This is paired with median filtering on the shape context and SIFT result sets based on matching distance to generate the final results. The ranking parameters for a result are below.

1. Semantic meaning of query sketch.
2. Shape context distance from query sketch s
3. SIFT distance from query sketch d

5.3.2 SVM Sketch Classifier

We have tweaked the open-source Caltech 101 classifier for classifying sketches. The classifier uses dense SIFT descriptors and spatial histograms, and in our case, it was trained on the sketch dataset provided by TU Berlin. As mentioned previously, this dataset has 250 categories each having 80 sketches in them. We use 15 sketches per category to train the classifier and 15 for testing. Figure 5.6 shows the training phase for svm sketch classifier. The trained model is stored in memory to use during the ranking phase.

The SVM classifier takes the sketch as input and returns the likely object categories with associated scores; we select the top three classes and filter the results based on these classes. Sketches in the database have already been given a label. Either it has been provided as a pre-determined one, or the sketch is given the label which the majority of its nearest neighbors have. Then, using the labels with the

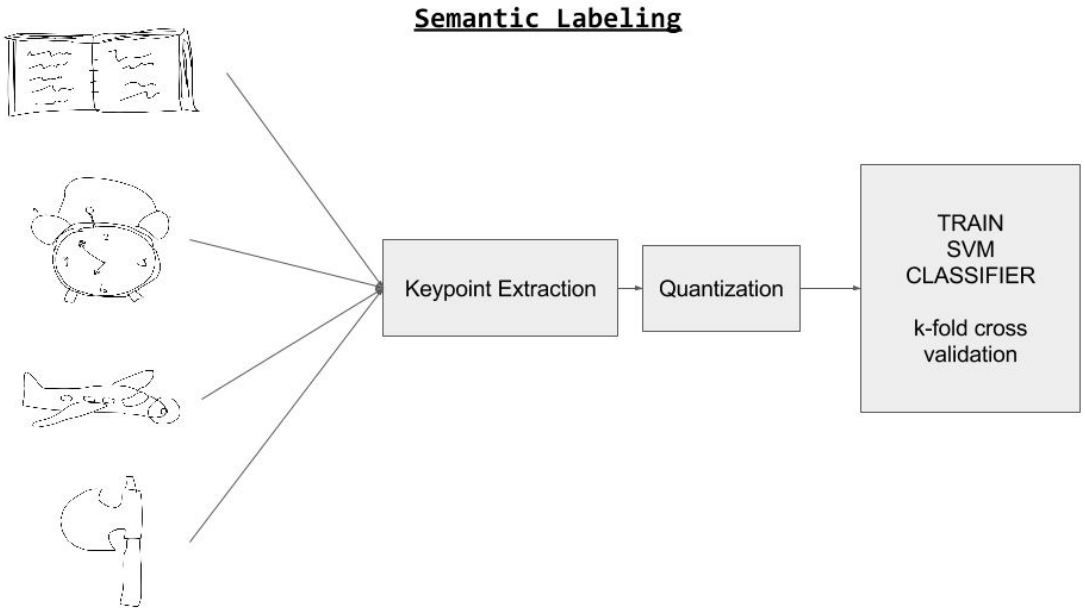

Figure 5.6: SVM classifier training

database sketches, if none of the labels output by the SVM match against a particular sketch, it is discarded from the result set. Predicting multiple-labels instead of one improves our overall retrieval quality as the classifier gets the correct semantic meaning in the top three labels with much higher accuracy than it would given only one label (which averages about 65% accuracy).

5.3.3 Median Filtering

Ranking is done in three steps. After the first stage, the semantic filtering using the SVM classifier described above, we examine the distances of the SIFT and shape context descriptors. These distances are stored in the metadata of each sketch in the result set. By taking the median distance, we remove all sketches in that result set (SIFT or shape context) that are outside the bound defined by that distance. This trims the result set to a much closer group of nearest neighbors. In the final step,

these result sets are merged by a union operator which yields the final set of results presented to the user. Figure 5.7 outlines the ranking pipeline.

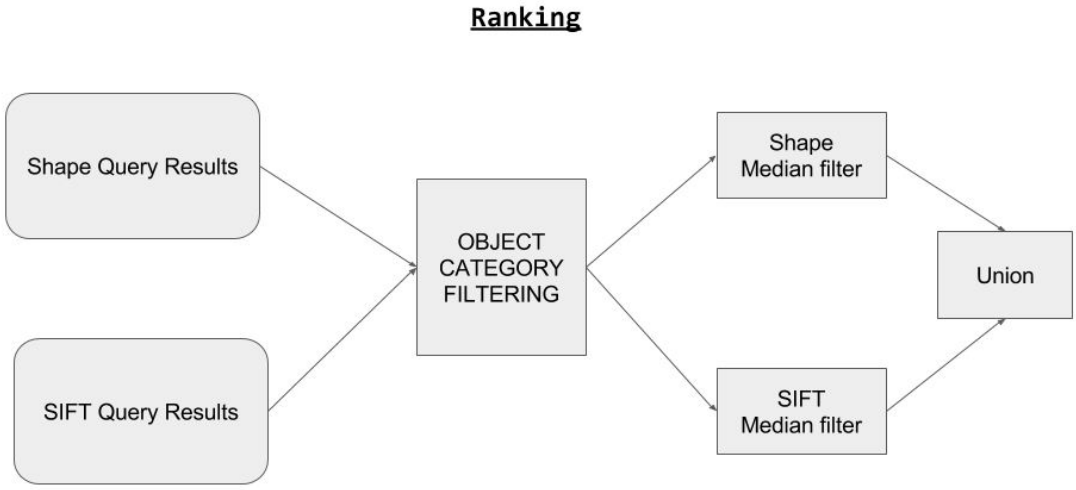

Figure 5.7: Ranker stage overview

6. RESULTS AND DISCUSSIONS

To measure the performance of our system, we evaluated in terms of both efficiency and sketch retrieval accuracy.

6.1 Efficiency

A sketch retrieval system working in real-time must have a low-latency sketch matching system. The basic unit of a sketch matching system is the average time taken to compute similarity between two sketches. We consider the latency of our system against those for three of the most popular matching methods used today: Hausdorff distance comparison, shape descriptor matching, and sift descriptor matching. Table 6.1 shows the average matching latency for all four of the methods tested.

Search Method	**Median**	**Average**	**Std Dev**
SketchSeeker	1.8	1.9	0.43
Hausdorff Retrieval	192.5	195	16
Uncompressed Shape Descriptors	290.2	297	18

Table 6.1: Latencies of each sketch matching metric. Note that Hausdorff and shape context descriptors must search through the entire database pairwise. Here we see the improvements provided by SketchSeeker over the pairwise-based matching.

The latency was computed as the average time across 100 sketch queries, including both simple and complex shapes. Note that while SketchSeeker can match against all sketches in the database in a single query, the other metrics match against each sketch in the database, leading to dramatically slower query times that are unsuitable for real-time systems.

A more interesting metric is comparing the latency of SketchSeeker for a complete database search against a single matching operation in the other metrics. Again, we compute the latency for the three other popular matching methods, but we only consider the time taken for determining one match. The results, shown in Table 6.2,

Method	**Median**	**Average**	**Std Dev**
Hausdorff Retrieval	502	514	15
Uncompressed Shape Descriptors	913	923	115
Uncompressed SIFT Descriptors	80	83	6

Table 6.2: Latencies of each sketch matching metric but only considering a single sketch match in the case of every method other than SketchSeeker.

show how SketchSeeker scales to an enormous database of sketches by performing a database-wide search in nearly the time other metrics take to consider a single pair.

These results show significant speed advantages to searching through large sketch databases using SketchSeeker, but there are also storage advantages. As mentioned before, SketchSeeker requires only a few numbers to represent a sketch along with a trained autoencoder network. The other methods require more storage space, or as in the case of Hausdorff, the entire sketch.

6.2 Retrieval Accuracy

We compare the results of a SketchSeeker retrieval against those generated by the Hausdorff distance metric and shape descriptors. To do this, we perform a query in SketchSeeker which will return a set of K nearest neighbors based on a threshold. We then use the same query sketch and scan through the entire database of sketches calculating the top K sketches based on Hausdorff distance and shape descriptors. This is a time-consuming process, as seen in the previous section on latencies, but

it approximates the output of SketchSeeker by finding nearest neighbors using only the Hausdorff or shape descriptor metrics.

The precision metric computed is based on the similarity of the result set to the input query and the associated class labels. Similar sketches of one of the three potential classes are considered as true positives, while sketches which are not similar and do not match the actual query sketch's label are considered as false positives. This enables us to generate a precision metric, although it is worth noting that we are not performing classification in this case, meaning other classification metrics will not be applicable.

As seen in Table 6.3, SketchSeeker attains a higher precision than the other metrics, especially Hausdorff. Hausdorff metric may work reasonably well on simple-shaped sketches, but the precision drastically goes down with complex sketches. Shape descriptor works well on sketches where outer shape is the most distinctive feature; it falters considerably in sketches where internal structure is important. SketchSeeker performs the best of all three in every case. This indicates the usefulness of considering both the outer shape and internal structure of the sketches when matching.

Search Method	**Median**	**Average**	**Std Dev**
SketchSeeker	87.5%	80%	25%
Hausdorff Retrieval	20%	25%	18%
SketchSeeker without SIFT keypoints matching	85%	74%	30%

Table 6.3: Precision of the result set for multiple metrics, including SketchSeeker.

Also, from Table 6.3, we see that the results suffer from a high standard deviation. This is due to the varying complexity of sketches. By looking at the entropy of the

sketches in this dataset, we see that some are very simple, perhaps needing only outer shape to identify them, while others are highly complex with a lot of internal structure. However, we see that the median precision is higher than the average for SketchSeeker. This indicates that while complex sketches may account for a lot of variation and be difficult to match, most of the time, sketches will be midway between simple and complex, and in these cases, SketchSeeker performs quite well. This analysis of the standard deviation is only interesting in regards to precision; we did not see the effects of complexity in the latency evaluation since the lookups require the same amount of time no matter the sketch content.

Further, consider the case of overtracing, which may often be the case for novice sketches. In Figure 6.2 we see the sensitivity of the shape descriptor to overtracing. Even though the general shape is fairly accurate around the edges, the extra strokes lead to sketches with more complexity. When the shape is relatively simple with no overtracing, Hausdorff and shape can be fairly accurate, although Hausdorff is not as sensitive to overtracing as seen in Figure 6.1. SketchSeeker performs well regardless, being largely insensitive to the limits of the other metrics like scale, outliers, and overtracing; see Figure 6.3. More results show that in the case of noisy sketches our approach works much better than both the other distance metrics as well. Note that in all query retrieval figures, the first sketch shown is always the query sketch.

In sketches having a rich inner structure, both Hausdorff and shape context descriptors fare poorly, whereas SketchSeeker again outperforms. See Figures 6.4, 6.5, 6.6, 6.7, and 6.14 for examples.

A complicated sketch category is 'arm'. It does not have a fixed shape and has features like muscles, fingers, and an elbow. Shape query retrieval precision for 'arm' sketches are low, but the SIFT query retrieval precision is high. Figure 6.8 shows a query with an arm. The predicted labels for this category are 'arm' and 'foot',

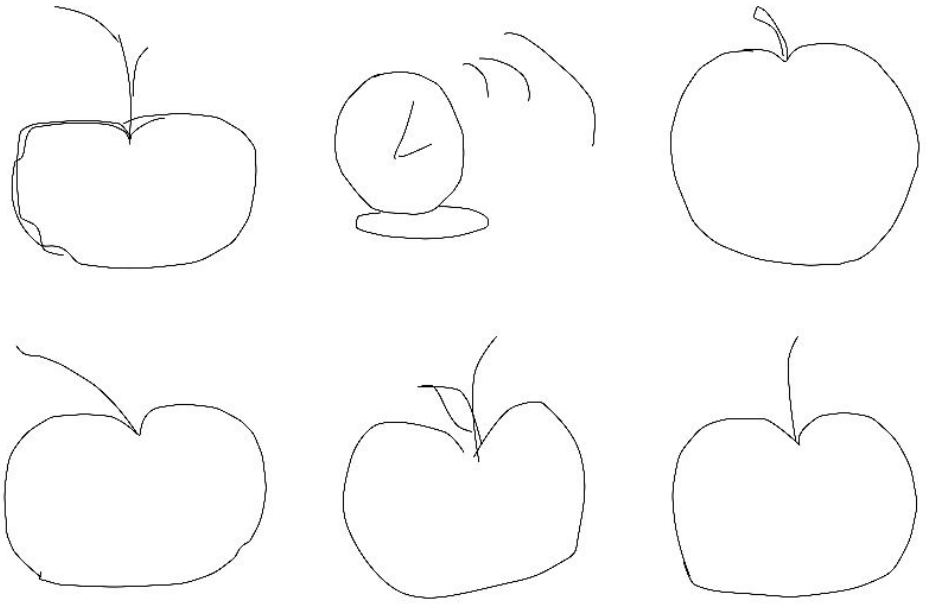

Figure 6.1: Apple sketch matched by Hausdorff metric.

again reminding us of interclass similarity. SketchSeeker also performs well in this category, showing it handles the interclass and intraclass complications better than the other methods. Figure 6.9 shows a query with a foot.

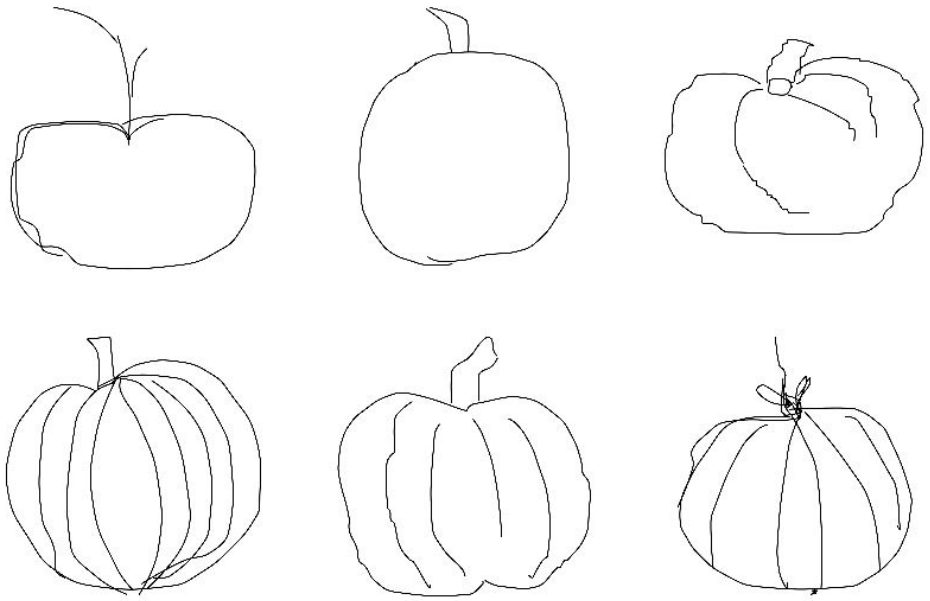

Figure 6.2: Apple sketch query retrieval by SketchSeeker using shape context descriptor alone; note the false positives due to shape context descriptor matching of overtraced strokes

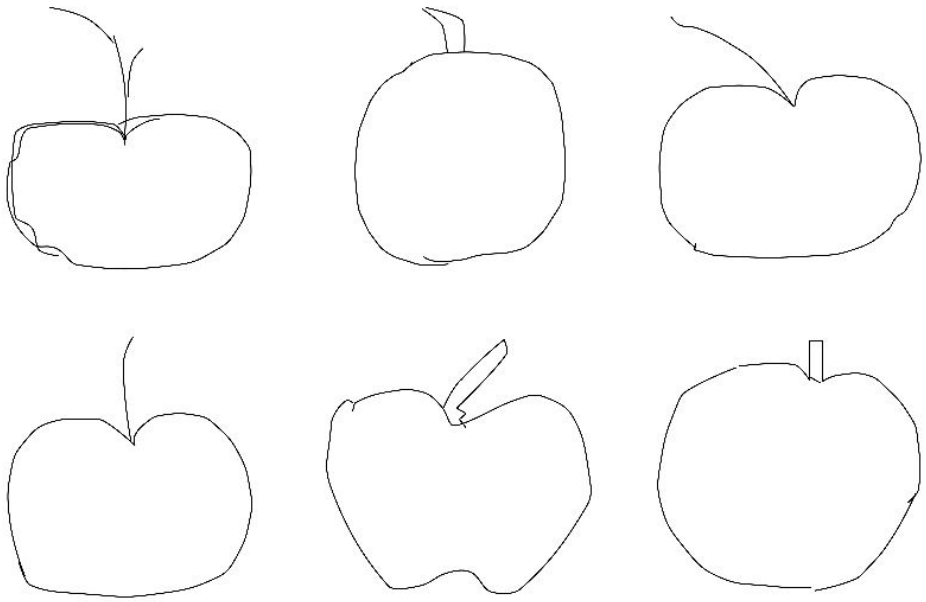

Figure 6.3: Apple sketch query retrieval by SketchSeeker.

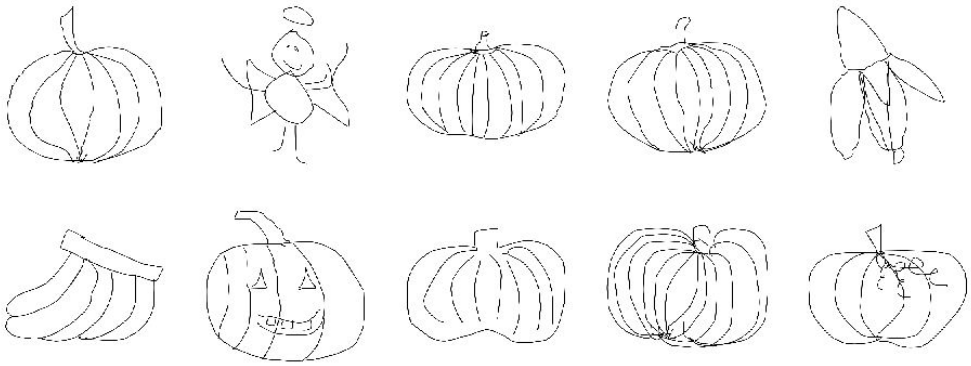

Figure 6.4: Results of pumpkin retrieval by SketchSeeker based on shape descriptors alone; note the false positives due to inner structure of pumpkins.

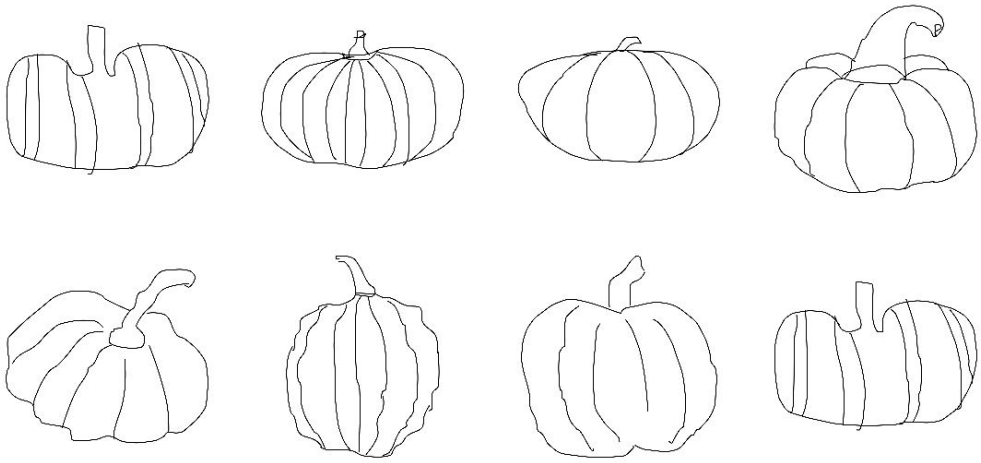

Figure 6.5: Results of pumpkin retrieval by SketchSeeker.

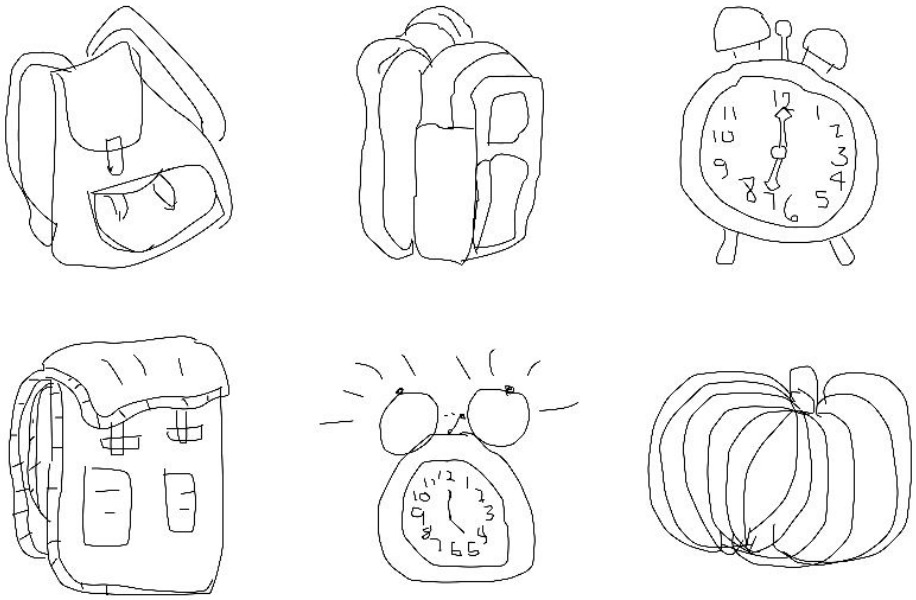

Figure 6.6: Results of bag sketch retrieval based on uncompressed shape descriptor matching alone

Figure 6.7: Results of bag sketch query retrieval by SketchSeeker; compare the results with uncompressed shape descriptor matching in Figure 6.6.

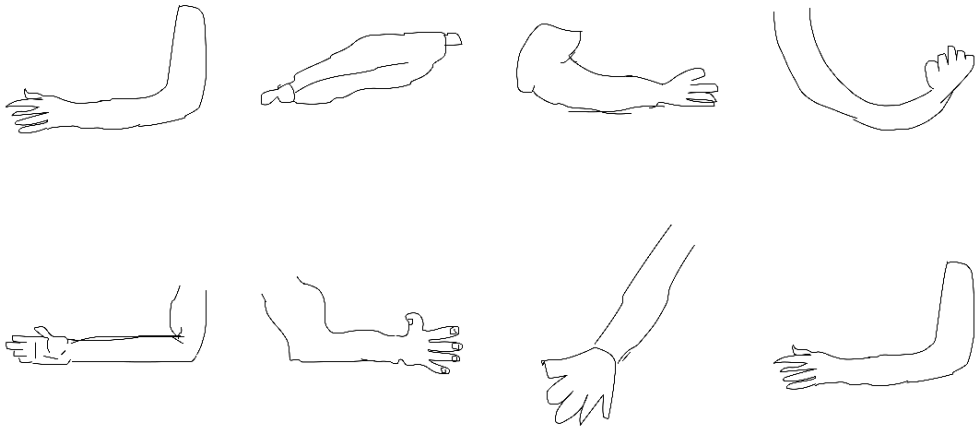

Figure 6.8: Arms sketch retrieval by SketchSeeker.

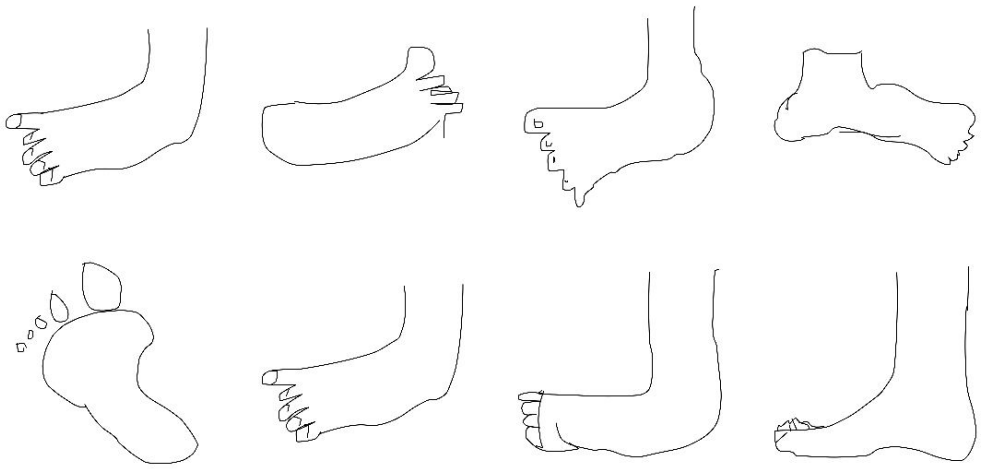

Figure 6.9: Foot sketch retrieval by SketchSeeker.

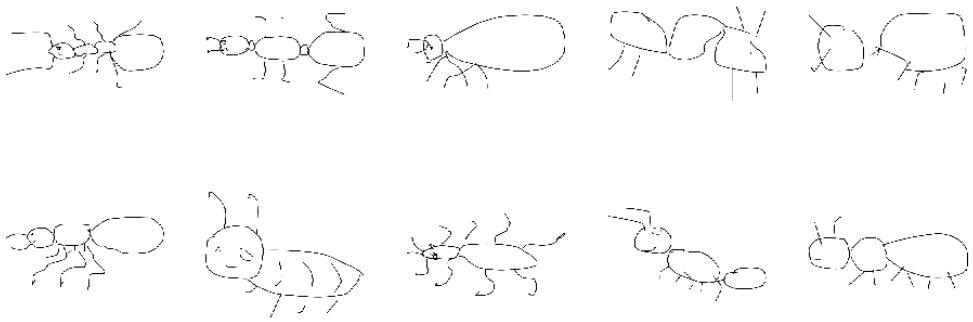

Figure 6.10: Ant sketch retrieval by SketchSeeker, another complicated category.

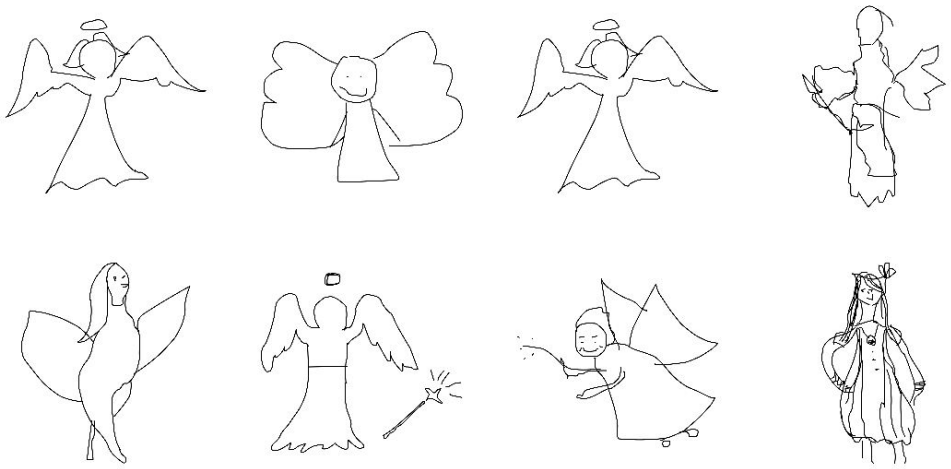

Figure 6.11: Angels sketch retrieval by SketchSeeker.

Figure 6.12: Bags sketch retrieval by SketchSeeker.

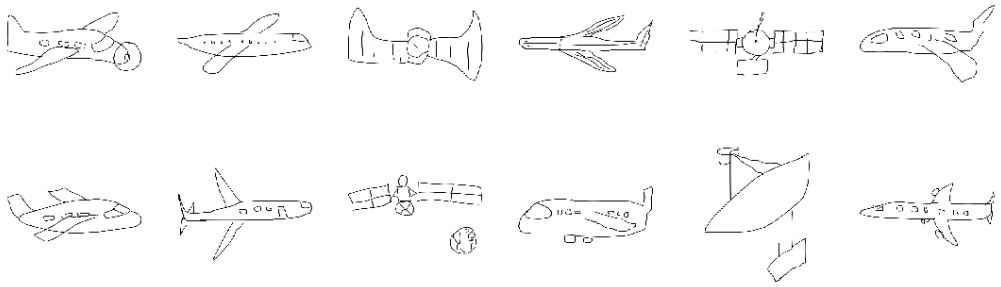

Figure 6.13: Airplanes sketch retrieval by SketchSeeker. Here we do see false positives in the result set, likely owing to the multiple components of the query sketch. For instance the rotor element of the sketch could introduce similarity to a shape like the antenna.

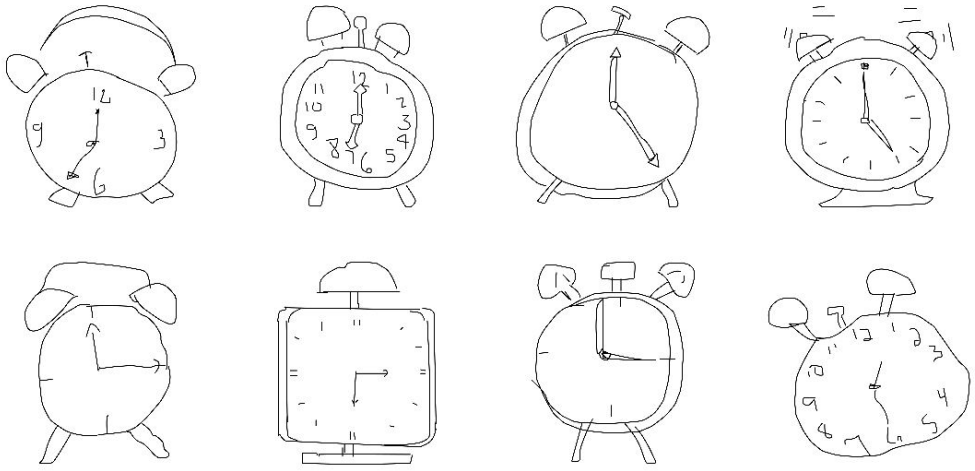

Figure 6.14: Alarm clock sketch retrieval by SketchSeeker.

7. FUTURE WORK

There are several interesting areas of research related to SketchSeeker which we would like to develop further. The first one is whether we can combine the global shape descriptor and local SIFT descriptor to come up with an entirely new descriptor for the index structure. The current indexing process is time consuming due to the long deep auto-encoder training phase. We would like to investigate some parallel approaches to speed it up. Also, we would like to investigate giving sketches a better semantic meaning than the one described in this thesis, since the SVM classifier is somewhat limited in its capabilities. Finally, we would like to better our confidence parameters by focusing more on the semantic meaning of sketches. In consideration of the broader field of sketch recognition, it is important to note that we have not explored the temporal information of a sketch's strokes in this thesis. Temporal information is widely used in sketch recognition application, and classification based temporally on strokes might lead to a feature in the matching framework in the future.

8. CONCLUSION

In this thesis, we described SketchSeeker, a sketch retrieval system for finding sketches similar to a given query sketch. We use both shape context descriptor and SIFT key-point descriptor in the matching framework. These sketch representations are then heavily compressed using deep auto-encoding and stored in a kD-tree for enormous improvements in storage and speed efficiency. Finally, we rank the result set retrieved for an input sketch by the semantic meaning of the query paired with median filtering on the distance of the matches to the query sketch.

Our approach towards sketch retrieval closely follows the design for any generic text or image retrieval system. The three subsystems described in our work – indexing, retrieval, and ranking – can be extended to design a domain-specific sketch recognition system which can be used for specific tasks like logo/trademark retrieval, engineering drawing retrieval, clipart finding, or multi-modal searching systems. Furthermore, the nearest neighbor search technique could be used to provide real-time suggestions for stroke completion to a user of a sketch-based interface. Overall, we show that SketchSeeker is a highly efficient sketch retrieval system in terms of time and space which obtains excellent similarity results for general input query sketches. With its compression, speed, and accuracy, it has many potential extensions for further application in specific sketch domains.

REFERENCES

[1] Olufunmilola Atilola, Stephanie Valentine, Hong-Hoe Kim, David Turner, Erin McTigue, Tracy Hammond, and Julie Linsey. Mechanix: A natural sketch interface tool for teaching truss analysis and free-body diagrams. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28:169–192, 5 2014.

[2] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *In NIPS*, pages 831–837, 2000.

[3] Serge Belongie, Greg Mori, and Jitendra Malik. Matching with shape contexts. In *Statistics and Analysis of Shapes*, pages 81–105. Springer, 2006.

[4] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[5] Tania Di Mascio, Marco Francesconi, Daniele Frigioni, and Laura Tarantino. Tuning a cbir system for vector images: The interface support. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 425–428, New York, NY, USA, 2004. ACM.

[6] Mathias Eitz and James Hays. Learning to classify human object sketches. In *SIGGRAPH 2011: Talks*, 2011.

[7] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[8] Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. An evaluation of descriptors for large-scale image retrieval from sketched feature lines. *Computers & Graphics*, 34(5):482–498, 2010.

[9] Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-based 3d shape retrieval. In *SIGGRAPH 2010: Talks*, 2010.

[10] Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa. Sketch-based image retrieval: Benchmark and bag-of-features descriptors. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1624–1636, 2011.

[11] Mathias Eitz, Ronald Richter, Tamy Boubekeur, Kristian Hildebrand, and Marc Alexa. Sketch-based shape retrieval. *ACM Trans. Graph.*, 31(4):31, 2012.

[12] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[13] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[14] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE, 1999.

[15] Huanfeng Ma and David Doermann. Adaptive hindi ocr using generalized hausdorff image comparison. 2(3):193–218, September 2003.

[16] Michael Oltmans. *Envisioning sketch recognition: a local feature based approach to recognizing informal sketches*. PhD thesis, Massachusetts Institute of Technology, 2007.

[17] Tom Y. Ouyang and Randall Davis. Chemink: A natural real-time recognition system for chemical drawings. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, pages 267–276, New York, NY, USA, 2011. ACM.

[18] Brandon Paulson and Tracy Hammond. Marqs: retrieving sketches learned from a single example using a dual-classifier. *Journal on Multimodal User Interfaces*, 2(1):3–11, 2008.

[19] Glauco Vitor Pedrosa, Solange Oliveira Rezende, and Agma Juci Machado Traina. Reducing the dimensionality of the sift descriptor and increasing its effectiveness and efficiency in image retrieval via bag-of-features. In *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*, WebMedia '12, pages 139–142, New York, NY, USA, 2012. ACM.

[20] M. Pontil and A. Verri. Support vector machines for 3d object recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(6):637–646, Jun 1998.

[21] Dean Rubine. *Specifying gestures by example*, volume 25. ACM, 1991.

[22] Marçal Rusiñol and Josep Lladós. Efficient logo retrieval through hashing shape context descriptors. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, DAS '10, pages 215–222, New York, NY, USA, 2010. ACM.

[23] Rosália G Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. *ACM Transactions on Graphics (TOG)*, 33(6):174, 2014.

[24] Rosália G Schneider and Tinne Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. *ACM Transactions on Graphics (TOG)*, 33(6):174, 2014.

[25] Zhenbang Sun, Changhu Wang, Liqing Zhang, and Lei Zhang 0001. Query-adaptive shape topic mining for hand-drawn sketch recognition. In Noboru Babaguchi, Kiyoharu Aizawa, John R. Smith, Shin'ichi Satoh, Thomas Plagemann, Xian-Sheng Hua, and Rong Yan, editors, *ACM Multimedia*, pages 519–528. ACM, 2012.

[26] Zhenbang Sun, Changhu Wang, Liqing Zhang, and Lei Zhang. Query-adaptive shape topic mining for hand-drawn sketch recognition. In *Proceedings of the 20th ACM International Conference on Multimedia*, MM '12, pages 519–528, New York, NY, USA, 2012. ACM.

[27] Stephanie Valentine, Francisco Vides, George Lucchese, David Turner, Hong hoe Kim, Wenzhe Li, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-based tutoring system for statics courses. *Innovative Applications of Artificial Intelligence*, 2012.

[28] Stephanie Valentine, Francisco Vides, George Lucchese, David Turner, Hong-hoe Kim, Wenzhe Li, Julie Linsey, and Tracy Hammond. Mechanix: a sketch-based tutoring and grading system for free-body diagrams. *AI Magazine*, 34(1):55, 2012.

[29] Laurens JP van der Maaten, Eric O Postma, and H Jaap van den Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10(1-41):66–71, 2009.

[30] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. 2008.

[31] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[32] Andrea Vedaldi and Brian Fulkerson. Vlfeat: an open and portable library of computer vision algorithms. In Alberto Del Bimbo, Shih-Fu Chang, and Arnold W. M. Smeulders, editors, *ACM Multimedia*, pages 1469–1472. ACM, 2010.

[33] Changhu Wang, Zhiwei Li, and Lei Zhang. Mindfinder: Image search by interactive sketching and tagging. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1309–1312, New York, NY, USA, 2010. ACM.

[34] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168, New York, NY, USA, 2007. ACM.

APPENDIX A

SIFT DESCRIPTOR

A SIFT feature [31] is a selected image region (also called keypoint) with an associated descriptor. Keypoints are extracted by the SIFT detector and their descriptors are computed by the SIFT descriptor. It is also common to use independently the SIFT detector (i.e. computing the keypoints without descriptors) or the SIFT descriptor (i.e. computing descriptors of custom keypoints). A SIFT descriptor is a of the image gradients in characterizing the appearance of a keypoint. The gradient at each pixel is regarded as a sample of a three-dimensional elementary feature vector, formed by the pixel location and the gradient orientation. Samples are weighed by the gradient norm and accumulated in a 3-D histogram h, which (up to normalization and clamping) forms the SIFT descriptor of the region. An additional Gaussian weighting function is applied to give less importance to gradients farther away from the keypoint center. Orientations are quantized into eight bins and the spatial coordinates into four each, as follows : Fig A.1

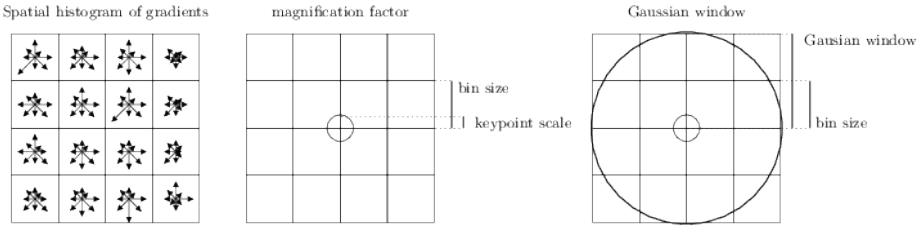

Figure A.1: SIFT bins [31]

APPENDIX B

SUPPORT VECTOR MACHINES

B.1 SVMs for classification

SVMs are well suited for general purpose pattern recognition. Given a set of points which can be linearly classified, a linear support vector machine finds the hyperplane leaving the largest possible fraction of points of the same class on the same side of plane while maximizing the distance of either class from hyperplane. SVM is thus called a maximal margin classifier. As a maximal margin classifier it is well suited for computer vision based recognition [20] [4] as the maximal margin minimizes the risk of misclassifying the instances in the training set but also the not yet seen instances of the test set. The high dimensionality of the problem space makes SVMs especially well suited for the problems.

B.2 SVM kernel trick

As wikipedia says, Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

B.3 Feature Extraction

We use SIFT features extracted from the sketches as the feature set for each training sketch.

APPENDIX C

DEEP AUTOENCODER

An autoencoder takes an input $x\epsilon[0,1]^d$ and first maps it to a hidden representation $y\epsilon[0,1]^d$. This is done through the help of an encoder. This latent representation y is now decoded back to a reconstruction z which is of the same shape as x. z is a reconstruction of x given a hidden representation y. The reconstruction error can be computed as mean squared error. The the code y is a distributed representation that captures the coordinates along the main factors of variation in the data x. If there is one linear hidden layer (the code) and the mean squared error is used as the reconstruction error and this criterion is used to train the network, then the k hidden units learn to project the input in the span of the first k principal components of the data. If the hidden layer is non-linear, the auto-encoder behaves differently from PCA, with the ability to capture multi-modal aspects of the input distribution. This property of capturing multi modal aspects make autoencoders far more powerful than PCA and this effect becomes pronounced when we consider stacking multiple encoders (and their corresponding decoders) when building a deep auto-encoder [13]. y the hidden layer representation of x is considered to be a lossy compression of x. A denoising autoencoder forces hidden layer to discover more robust hidden features by trying to learn and reconstruct the input from a corrupted version of it. The denoising auto-encoder is a stochastic version of the auto-encoder. A denoising auto-encoder does two things, encode the input and try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder.

C.1 Introduction

A deep autoencoder is a multilayer neural network which is typically composed of two symmetrical deep belief networks that have a few layers representing the encoding half of the net, and the second set of layers making up the decoding half. The layers are restricted Boltzmann machines, the building blocks of deep belief networks.

C.2 Architecture

C.2.1 Encoding

An encoder in a deep auto-encoder will have multiple layers. The compressed feature vector y represents the distributed representation of input x.

C.3 Decoding

The second symmetrical deep belief network is that of a decoder. It is the part that learns to reconstruct the input. It does so with a second feed-forward net which also conducts back propagation. The back propagation happens through reconstruction error.

C.4 Hashing and Compression

Let the compressed feature vector y be of size n i.e the net produces a vector if n numbers after encoding. This n-number vector is the last layer of the first half of the deep autoencoder, the pre-training half, and it is the product of a normal Restricted Boltzmann machine. These n numbers are the compressed representation of input data. It can also be used as similarity preserving hash code. The similarity between hash codes gives the similarity between input data they represent.